A Multi-View Tool for Checking the Security Semantics of Router Configurations

Dr. Holger Peine, Dr. Reinhard Schwarz
Fraunhofer Institute for Experimental Software Engineering, Germany
{peine, schwarz}@iese.fraunhofer.de

Abstract

Routers are critical components of IP networks, but hardly any tool support for analyzing their security exists to date. We have developed such a tool, named CROCODILE, that tracks the security implications of related configuration directives that may be scattered all over the router's configuration, instead of analyzing only isolated configuration clauses like other tools do. Our tool offers several novel evaluation capabilities and presents its findings as a collection of multi-view displays, enabling the user to focus on selected aspects, and to navigate deeper and deeper into specific details. We demonstrate the practical use of CROCODILE, and a comparison with the well-known RAT tool illustrates CROCODILE's remarkable capabilities.

1. Router configuration and security checking

Today, all but the smallest enterprises are connected to the Internet and use IP networks internally, too. Routers play a fundamental role in such networks as they relay (or deliberately do not relay) IP packets between source and destination hosts. Modern routers do not only perform relaying functions, but also filtering, separation, encryption and monitoring of data streams. Furthermore, they provide various management interfaces for configuration, (remote) maintenance, and monitoring. All these functions potentially affect the availability, integrity, and confidentiality of data connections, thus making routers highly security-critical network components.

However, configuring a router is a difficult and error-prone task. The available configuration languages are complex and often badly documented, and errors are hard to spot. Therefore, tool support to uncover hidden vulnerabilities in router configurations is highly welcome, as tools improve efficiency and effectiveness as well as objectivity and repeatability of router security checks. However, few such tools exist to date, and the existing ones work at a rather low level of abstraction and with very limited evaluation capabilities.

1.1. A tool for analyzing router security

CROCODILE[1] is a router security checker we have developed with support by Deutsche Telekom [1]. It parses a router configuration file (i.e., a text file with a sequence of configuration clauses), checks for potential vulnerabilities, and generates rich HTML evaluation reports. Many trivial errors and inconsistencies that would easily escape the human eye are uncovered, none of which would be flagged by the router itself; however, CROCODILE's capabilities go far beyond such purely syntactic analysis:

- CROCODILE performs a *semantic analysis* of the configuration including, for example, the order or the omission of clauses, and their meaning in context. This is only possible because the tool analyzes the configuration as a whole, instead of each line in isolation.
- The often confusing wealth of evaluation results is presented using various *task-specific displays*. For example, the tool offers an annotated overview, a differential display relative to an earlier version of the configuration (see Section 3.3), detailed analyses, statistical data, and convenient hyperlinks to vendor documentation. A particularly useful and innovative detail analysis is the computation of the sets of packets effectively accepted ("whiteset") or rejected ("blackset") by the access control list protecting a router interface (see Section 3.5).
- Findings are logically grouped to various *analysis views*, with a view gathering all findings conceptually relating to a certain configuration aspect as defined by the user (e.g., authentication or logging). The clauses

1. "Cisco ROuter COnfiguration DILigent Evaluator"

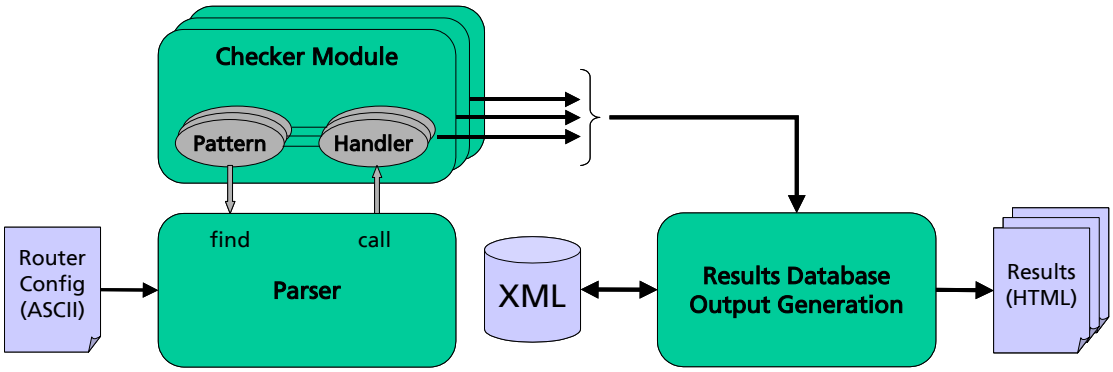

Figure 1. Structure and operation of CROCODILE

covered by a view may be scattered across the configuration, and one clause may contribute to several views. Findings may also be attached to a view as a whole. During inspection, the user may focus on any particular view. The view-based presentation makes the reasoning behind the router configuration easier to conceive, and consequently makes errors and vulnerabilities more transparent.

CROCODILE thus relieves the user from cumbersome low-level work and raises router configuration to a semantic level where the human expert is more adequately supported. None of the router security tools known to us has comparable capabilities.

1.2. Cisco Routers and IOS

CROCODILE is currently geared towards routers from Cisco Systems, the market leader, running the IOS operating system [2]. However, thanks to the modular design of the tool framework, vendor-specific modules could easily be replaced in order to support any text-based configuration format, even including types of devices other than routers.

Correct use of IOS poses a difficult task to the network administrator. The options provided by this operating system are numerous and complicated, and new IOS versions appear frequently. Furthermore, the syntax of IOS clauses is not very expressive, hardly intuitive, occasionally ambiguous, and often poorly documented. Finally, IOS further increases the administrator's burden by silently ignoring erroneous clauses instead of printing an error message. These adversities have greatly complicated the development of CROCODILE, but the user can now profit from the effort expended in developing the tool, and the knowledge embodied therein.

2. CROCODILE architecture and operation

Besides the run-time functionality of the logical views and the task-specific displays, CROCODILE is an extensible framework providing all functionality useful for general text-based security checking, while encapsulating specific areas of security evaluation in independent checker modules. Further design goals include the configurability of the checker modules in order to adapt to local policies, and portability, realized by implementing CROCODILE in Perl[1] without using any additional libraries.

2.1. Components

The tool is basically composed of three types of components: A pattern matching module (parser), an arbitrary number of checker modules, and a database and display module. Each *checker module* covers one logical aspect of router security with all the required checking logic. The *parser* provides the checker modules with input data relevant for the module's analyses. The *display module* generates complex but convenient hypertext documents from the findings. Figure 1 shows the interplay of these components.

2.2. Operation

The tool's basic operation is as follows: The parser initially polls all configured checker modules, retrieving from each module a set of *text patterns* this module is interested in. These patterns are described in Backus-Naur-Form, comparable to the format that appears in the Cisco documentation [2], providing the usual operators of sequence, alternative, repetition, and negation. Pattern macros may be defined for

1. Perl is freely available for all common operating systems (see, e.g., http://www.perl.org).

convenience. (An example of such pattern and macro specifications is shown in Figure 3 in Section 3.4.1).

The parser then proceeds to read an IOS router configuration file in text format, parses it line by line, and tries to find lines matching one of the patterns registered by the modules. Whenever a matching pattern is found in the configuration, the parser notifies all checker modules that had registered this pattern, and supplies the matching line in various formats to each module.

When a module receives such a notification, it executes a *pattern handler* associated with this pattern. Such a handler is a method of the checker module that contains all checking and evaluation steps to be performed on each occurrence of this pattern. This may include using information gained from previous handler invocations (even handlers in other modules), thus building a comprehensive view spanning more than one line.

The invocation of each handler returns a checking result, and possibly additional comments. All such findings are stored in structured format in an internal database.

After all lines of the input file have been processed in this way, the parser polls each module once more for its overall conclusions. The polling enables the modules to construct an integrated view of the aspects interesting to them from the sum of all previous parser notifications.

After the parser run is finished, the complete result database is stored as a file in XML format. These raw data are then used to generate various integrated HTML hypertext reports viewable with any conventional web browser.

CROCODILE is currently delivered with a collection of standard checker modules that cover fundamental security aspects of router configurations. These modules will be described in Section 3.4.

2.3. Construction of custom checker modules

Users do not normally come into contact with the interface between the parser and the checker modules, even when they construct their own checker modules. The interplay results automatically, as all checker modules are derived from a common base class `Module` which provides all the machinery to interface with the rest of the system. Constructing a customized checker module thus more or less reduces to only three tasks:

1. Specify the patterns to be analyzed in Backus-Naur format.

2. Write one handler per pattern to analyze the text fragments found and reported by the parser.

3. [optional] Write a postprocessing handler to draw the overall conclusions from all findings of this module.

2.4. Result database

All checker modules store their results in structured format in an internal result database. The most important concepts in this respect are annotations and views.

Annotations are comments about a certain property of the router configuration. Every annotation bears a severity tag, with tag values ranging from `OKAY`, `INFO`, `CHECK`, `WARN` to `ALERT`. This tag indicates whether the annotation refers to a positive, neutral, unclear, negative, or even critical finding. The severity of a finding is indicated in the hypertext output by a corresponding color.

Views gather all findings referring to a common aspect of the configuration. The handler programmer may freely define the name and meaning of a view and assign every finding to one or more views. Views help to focus subsequent result display on selected aspects. Examples for common views include 'User Authentication', 'Logging', or 'Accounting'.

Besides annotations and views, *suggestions* (e.g., fixes), *references* to other findings or external resources, and *data dumps* may be stored in the database. A simple programming interface for all these result types is available to the handler programmer. Each such result item may pertain either to a line, an IOS configuration mode[1], a logical view, or to the configuration as a whole.

3. Practical Use

CROCODILE is very easy to install: The software archive file is unpacked to an arbitrary directory, and may be directly started from there, provided that a Perl run-time environment is present. All output generated by CROCODILE is stored below the installation directory, and the tool can be removed simply by deleting the installation directory. CROCODILE is invoked by supplying the name of an IOS router configuration file, and produces its output without further user interaction as a subdirectory of HTML files ready for viewing.

3.1. Task-specific displays

CROCODILE provides different displays of the evaluation results including overviews, statistics, and topic-oriented perspectives in the form of the views described above. The displays are enriched by hyperlinks to related displays and supplementary information such as automatically generated Internet links to corresponding entries of the Cisco IOS Com-

1. A configuration mode is a scoping concept of the IOS command shell, gathering a number of IOS commands all pertaining to, for example, a specific router interface ("interface mode") or serial line ("line mode").

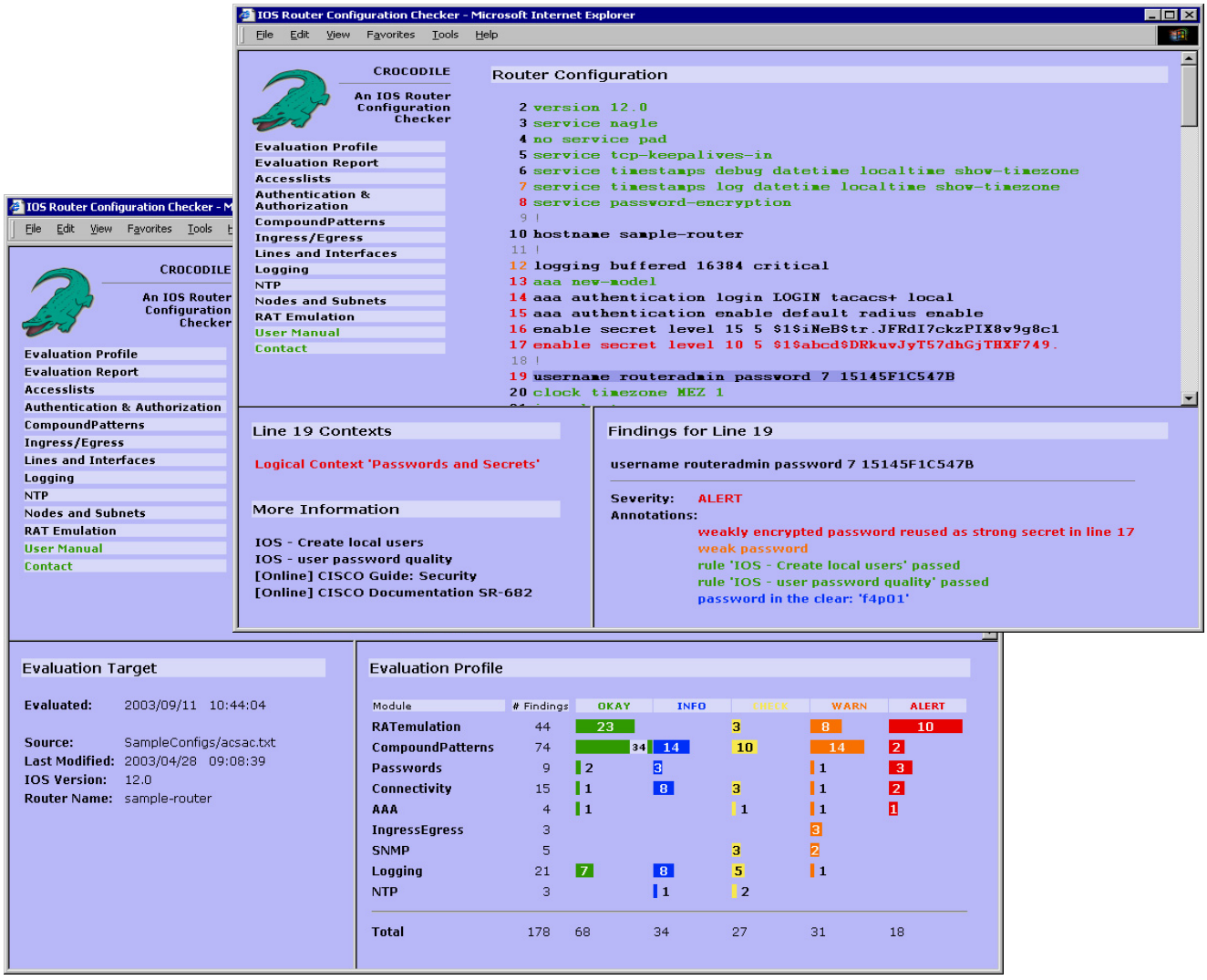

Figure 2. Two CROCODILE displays at different levels of detail

mand Reference. Starting from an overview display, the user can navigate deeper and deeper into specific aspects, with the displayed scope shrinking and the level of detail increasing. The user may focus on a particular topic by selecting the corresponding view. Coloring is used to indicate the severity assigned to each finding. Figure 2 shows two typical examples of such displays, with the one in the background pertaining to the configuration as a whole, and the upper one focussing on a specific configuration line (line 19 in this case).

3.2. Typical defects uncovered

Feeding the tool with the router configuration file that is available for an online demonstration on the CROCODILE homepage, our checker found, among others, the following exemplary problems:

- potentially dangerous services that have been left activated, and missed opportunities to activate useful additional security functions (e.g., password protection, encryption, authentication)
- strongly encrypted passwords that are reused in weakly encrypted format (thus providing almost plaintext for the strongly encrypted secret)
- incomplete configuration of complex router features (e.g., authentication based on RADIUS, but with no RADIUS server being specified)
- an assignment of an undefined access control list to an interface (leaving the interface silently unprotected)

- an access control list that is defined but never used (probably due to a typing mistake: chances are that the misspelled access control list actually exists but provides inadequate protection)
- access control lists that contain superfluous ("dead") rules (not an immediate danger, but an indication that the router administrator did not fully comprehend the implications of the configured properties) — see Section 3.5
- an ingress/egress interface (i.e., a connection to a foreign, potentially hostile network) that fails to satisfy the user-defined minimum relaying (whitelist) and filtering (blacklist) requirements — see Section 3.5
- a line interface failing user-defined blacklist and whitelist restrictions
- an inadequately high reporting threshold for logging (thus excluding access control list events that are explicitly required to be logged).

Note that most of these findings require an evaluation of the combined effects of multiple configuration clauses. This type of analysis cannot be achieved by mere pattern matching — i.e., by checking the structural equivalence to predefined clause templates at the syntax level — but requires an assessment of the semantics of the configuration as a whole.

Figure 2 shows the evaluation profile of a demonstration run. For more details about the configuration under test there and the results of its evaluation, we refer the reader to the CROCODILE homepage at `www.iese.fraunhofer.de/crocodile`.

3.3. Differential display mode

A special type of reporting is the differential display of *two* analysis results. Differential display mode emphasizes the effects of changes relative to an earlier configuration — all findings not affected by these changes are faded out in the display. This is very useful in practice, as the user will usually not "fix" all findings, but leave some settings as they are for reasons not visible in the configuration file. It would be a considerable nuisance to be presented the same irrelevant findings again and again. CROCODILE can restrict its evaluation to only the differences between two versions of a configuration, producing an evaluation profile display and a side-by-side overview display that make immediately obvious where the configuration was changed and whether the change was for the better or for the worse.

3.4. Available checker modules

CROCODILE currently comes with the following checker modules:

3.4.1. CompoundPatterns. The most fundamental checker module delivered with CROCODILE is the `CompoundPatterns` module. It supports user-defined evaluation criteria for the (non-)existence of certain IOS configuration clauses. The user may specify arbitrary syntax patterns and, for each pattern, assign a severity to (1) the occurrence of this pattern in the configuration and (2) the omission of the pattern. Depending on the user's choice, CROCODILE will generate an `OKAY`, `INFO`, `CHECK`, `WARN`, or `ALERT` message in each case. A pattern may be either a basic pattern referring to a single configuration clause, or a compound consisting of several basic patterns linked, for example, by logical AND, OR, XOR, NOT, or IF-THEN-ELSE operators. `CompoundPatterns` recognizes the IOS configuration modes (e.g., 'interface mode', 'line mode', 'router mode'), and patterns may be restricted to be valid in certain IOS modes only. `CompoundPatterns` presents its findings in a CROCODILE view of the same name.

Figure 3 shows an example of typical macro and compound pattern specifications. The first compound pattern requires that for all interfaces except 'Loopback' or 'Null' interfaces, the interface must be either disabled ("shutdown"), or else the Cisco Discovery Protocol (CDP) must be disabled — either local to the interface context ("no cdp enable"), or at global level for all interfaces("no cdp run"). If this condition is met, CROCODILE generates an `OKAY` message ('o'); otherwise, a `WARN` message ('w') is issued.

Of course this simple specification format covers only those evaluation criteria that can be recognized without extracting and accumulating state information from the configuration clauses that match the compound pattern. Typical evaluation problems covered by `CompoundPatterns` are, for example, checking that (un)desired services and modes are explicitly (de)activated, that certain commands are assigned the required level of privilege, or that passwords are assigned and sufficiently securely encrypted.

```
INTERFACE ::=   { int | interface }

INTERFACE !Loopback !Null ... {
 ow  if (NOT shutdown) {
       no cdp enable | Global(no cdp run)
     }
} INTERFACE ...

line Vty ... {
 ow  no exec | access-class NUM in
 oc  logging synchronous
} line ...
```

Figure 3. Compound pattern specifications

Relationships that require comparisons between specific attributes of multiple pattern instances cannot be expressed in compound pattern syntax, for example, that an arbitrary name found in use has been properly defined elsewhere. However, this deliberate restriction yields a very intuitive way to describe many simple properties of a configuration. Our experience has shown that `CompoundPatterns` contributes significantly to a large evaluation coverage. Nevertheless, an analysis of the complex aspects of a router configuration needs to dig deeper than this. To this end, CROCODILE provides the more specific modules described next, which offer advanced capabilities at the expense of less flexibility in module configuration.

3.4.2. Connectivity. This module extracts from the configuration information about the topology of the router's network neighborhood with its surrounding subnets and nodes. While it does report irregularities and potential weaknesses, the focus of this module is not on security checking, but on extracting and deriving topology information for later processing by other modules or for manual inspection. Currently, the reconstructed topology data are listed in textual form, but a graphical output format for human users is under development. Taking the case of an interface as an example of the extracted information, the module would list which IP protocols are routed through this interface as inbound and outbound traffic to and from which subnets or hosts. `Connectivity` presents most of its findings in CROCODILE views for 'Lines and Interfaces' and 'Nodes and Subnets' .

3.4.3. IngressEgress. This module verifies that the configuration of access control lists (ACLs), line interfaces, and network interfaces conforms to the desired behavior as specified by the CROCODILE user. The user may specify blacksets and whitesets for these objects, and the `IngressEgress` module will list any violations found in the ACL relative to these sets, as well as perform other analyses as described in Section 3.5. Figure 4 shows an example of a user-defined egress restriction. Note that format and order of independent egress clauses is irrelevant: `IngressEgress` computes the precise meaning — i.e., blackset and whiteset — of the specification, and checks whether the evaluation target is at least as restrictive as the blackset, and open at least for the whiteset. The module presents its findings in various CROCODILE views for 'Ingress/Egress' in general, for 'Accesslists', and for 'Lines and Interfaces'.

3.4.4. AAA. The authentication/authorization/accounting (AAA) functionality of IOS is central to securing access to a router. The `AAA` module checks the consistency, completeness and security of the authentication and authorization settings (accounting is not yet assessed), including the order of clauses, the consistency of definition and use of mechanisms,

```
INGRESS_EGRESS_INTERFACE ::= Serial {1|2}
NETMASK  ::= 0.0.0.255
LOCALNET ::= 157.106.153.0 NETMASK

!  RFC1918 outbound address filtering

interface INGRESS_EGRESS_INTERFACE out {
   deny   ip any 10.0.0.0   0.255.255.255
   deny   ip any 127.0.0.0  0.255.255.255
   deny   ip any 172.16.0.0 0.15.255.255
   deny   ip any 192.168.0. 0 0.0.255.255
   permit ip LOCALNET       any
}
```

Figure 4. IngressEgress specification

the use of insecure or deprecated features, the use of IOS commands that are incompatible with AAA, as well as use and security of passwords and of external authentication server accesses. The evaluation of AAA is a complex task, and numerous non-AAA features like interface, line, user, and server commands have to be taken into account for this analysis. The module feeds its findings into the view 'AAA' of the 'Authentication & Authorization' display.

3.4.5. Logging. IOS supports five modes of logging relevant events at different severity levels (e.g., console logging, syslog, SNMP traps), which may be used in any combination. The `Logging` module checks and lists the activation level of logging modes, remote logging servers involved, and configured attributes such as time stamps, bandwidth limitations, or buffer sizes. The module validates the consistency of all these settings. `Logging` collects its findings in a CROCODILE view of the same name.

3.4.6. Passwords. Several passwords may be specified for an IOS router, such as login passwords, enable passwords, or FTP passwords, optionally restricted to certain IOS "enabling levels" only. Each password may be encrypted using either a cryptographically weak or strong mechanism. The `Passwords` module checks and lists the general and level-specific activation of passwords, checks for trivial or weakly encrypted passwords, and whether strongly encrypted passwords are reused in weakly encrypted format. The module presents its findings in the 'Passwords and Secrets' view of the 'Authentication & Authorization' display.

3.4.7. SNMP. The Simple Network Management Protocol (SNMP) is a protocol for the remote retrieval and manipulation of numerous router parameters pertaining to network

management. In particular SNMPv3 supports multiple mechanisms for authentication, authorization, and accounting. CROCODILE checks the router for SNMP issues like full use of the available security features, easily guessed community strings, protection by access control lists and SNMP view restrictions, and (de)activation of certain SNMP traps (i.e., asynchronous notification messages). `SNMP` collects its findings in a CROCODILE view of the same name.

3.4.8. NTP. The Network Time Protocol (NTP) supports the synchronization of system clocks among network components, which is useful for monitoring and error handling. Among other things, the `NTP` checker module evaluates NTP access control lists, checks and lists the declared and used authentication keys, and performs cross-checks between the configured NTP commands and these security settings. Furthermore, the potential NTP client hosts are listed for each router interface. `NTP` collects its findings in a CROCODILE view of the same name.

3.4.9. RATemulation. Another module, which is somewhat out of line with the others in that it does not cover a security aspect of its own, is a module that emulates the Router Audit Tool (RAT). Section 4.1 describes RAT and our emulation module in more detail.

3.5. Access control list analysis

A unique feature of CROCODILE is the symbolic representation and processing of access control lists (ACLs) and their associated blacksets and whitesets, that is, the sets of packets that can (not) pass through an ACL. In IOS an ACL is specified as an ordered sequence of 'permit' and 'deny' rules. Each rule refers to packets with a specific combination of protocol type (e.g., IP, TCP, UDP, ICMP, IGMP), origin address, destination address, and maybe some additional packet attributes. Conceptually, the router applies an ACL to a given IP packet by sequentially comparing the packet to each ACL rule; the first rule that matches the packet determines whether the packet is accepted (’permit’ rule) or rejected (’deny’ rule). Obviously, the order of rules is significant here. If none of the ACL’s rules matches the packet, the packet is silently discarded. This ACL formalism is simple yet flexible.

Unfortunately, ACLs tend to grow over time, and the growth is often rather unsystematical. Therefore, it is often infeasible for a human to determine the exact whiteset and blackset of a real-world ACL, and thus errors in the filtering behavior of a router are easily missed. Note, for example, that if an ACL rule appears too late in the rule sequence it may be unreachable for any matching IP packet — the rule may unintentionally become a dead clause. Note also that an intended ACL effect may be achieved in many different, syntactically unrelated ways, which makes two ACLs virtually incomparable for the human expert without knowing their blacksets and whitesets. CROCODILE computes blacksets and whitesets independently of a specific ordering or format of ACL and routing clauses, and uses them for a number of consistency and security checks:

- ACL rules whose packet domain intersects with that of earlier rules must be reconsidered when removing or reordering ACL clauses. CROCODILE indicates such dependent rules along with the set of rules they depend on.
- Rules which could be removed without any effect (dead rules) are flagged, along with the set of earlier rules whose combined effect "killed" them by "shading" the rule’s whole packet domain. Such a dead rule is a hint that the creator of the ACL originally intended a different effect than that actually resulting.
- The `IngressEgress` checker module (see Section 3.4.3) employs the blackset and whiteset computation when verifying that a given ACL (or, more generally, an interface specification in combination with associated routes and ACLs) does indeed amount to (at least) the relaying and filtering effect that the user stipulated.
- For a complex ACL with many rules, the corresponding blackset and whiteset may become so fragmented that manual inspection of all fragments is no longer feasible. CROCODILE’s 'blackwhite.pl' utility can be used in such cases to interactively narrow the scope of the set computations from full generality (i.e., all IP packets) to arbitrary subsets of IP packets — for example to "ICMP packets originating from remote addresses" or "TCP packets belonging to not-yet-established connections" only: Such blackset and whiteset projections are much easier to comprehend and to verify.

4. Related Work

Few tools are available for router security checking, none of them, remarkably, by the router vendors. The well-known RAT (Router Audit Tool) [3] and one lesser known tool by Equant Communications [4] are the only serious contenders known to us. Also technically interesting are some tools aimed at firewalls rather than routers, which analyze packet filtering behavior in a way that is comparable to CROCODILE’s functions in this area.

4.1. RAT

RAT version 2.0 is a joint development of several partners, including the U.S. National Security Agency (NSA), UUNET, and the SANS Institute. Like CROCODILE, RAT

is implemented in the Perl programming language. The tool is freely available from the Center for Internet Security [3].

RAT checks router configurations against configurable checking rules mirroring NSA recommendations [5]. A RAT rule consists of a pattern, which is declared as either required or forbidden. The rule is considered a `PASS` or `FAIL` depending on whether or not the corresponding pattern appears in the configuration text.

RAT's evaluation result is a list of all checking rules, each marked either as passed or as failed. Checking rules may be enriched by attributes such as a rule weight reflecting their importance, plaintext descriptions and explanations, and possibly a fix suggestion — that is, a sequence of IOS commands for correcting the failed configuration clause. These data are added to the evaluation report in the form of hyperlinks and support the user in understanding the findings. In CROCODILE terms, RAT thus offers three types of displays: (1) weighted `PASS/FAIL` statistics; (2) an explanation of the checking rules; (3) a sketch of possible corrections.

Similar to CROCODILE, RAT patterns may also be specified relative to certain IOS configuration modes. For example, they may be restricted to certain interface or line contexts. However, RAT does not support individual pattern handlers.[1] And in contrast to CROCODILE's compound patterns, RAT is essentially restricted to basic patterns. That is, evaluation criteria based on the interplay of multiple unrelated configuration lines cannot be handled with RAT's simple pattern-matching approach.

RAT thus lacks the abilities for analyzing the deeper semantics of several related, but scattered IOS configuration clauses. This becomes clear when taking a closer look at the test run discussed in Section 3.2 that included a standard RAT evaluation[2], depicted in Figure 2:

- In the example shown, line 19 passed the RAT rule "IOS – user password quality". In fact, in the local context of line 19 the password 'f4p01' is reasonably secure. The reason why this password should be rejected is because it is identical to the strongly encrypted enable secret in line 17 and only weakly encrypted, thus compromising the strong encryption by pointing a potential password cracker to the required plaintext.
- Another typical weakness of the RAT approach shows in ingress/egress checking. The RAT distribution contains a rule "IOS – ingress filter definition" that tries to check for RFC2827-compliant ingress filtering [6]. This rule, however, performs simple pattern matching of an ACL template containing the required clauses in predefined ordering and format. Consequently, RAT will report a "false positive" (i.e., a wrong `FAIL` decision) if the configuration happens to contain the required IOS clauses, but in a different order, or if it contains different clauses that nevertheless amount to equivalent filtering. RAT may even report a "false negative" (i.e., a wrong `PASS` decision) if the required sequence of ACL rules occurs in the configuration, but is made unreachable by a preceding, conflicting rule such as "`permit ip any any`" — a worst-case example. RAT cannot recognize this type of problems, because RAT rules only assess the structure, not the meaning of an IOS configuration.

These shortcomings come as no surprise when noting that the analysis capabilities of RAT correspond exactly to those of CROCODILE's `CompoundPatterns` checker (restricted to only basic patterns without any compounds)— a versatile, but rather shallow module. Admittedly, `CompoundPatterns` refrains from attributing its rules and delivering further explanations about the interpretation of its findings (which other modules certainly do); however, this restriction was deliberately chosen for this particular module in order to keep the effort for specifying user-defined evaluation criteria at a minimum, and the rule syntax simple.

On the whole, RAT compared to CROCODILE offers relatively weak, but richly documented checking rules. In fact, CROCODILE is able to benefit from RAT's elaborated rule sets, in particular from the links to further reading that are embedded in these rule specifications: A special checker module — `RATemulation` — takes RAT rule descriptions as input and faithfully applies the specified evaluation criteria to the IOS configuration under test. In addition to customary CROCODILE views, which (among other findings) contain the extracted RAT links, the emulation module produces reports equivalent to those of original RAT. This offers a migration path for former RAT users and also demonstrates the superiority of the CROCODILE framework. Note, however, that the available RAT checking rules do not improve CROCODILE's depth of analysis, but merely complement the "More Information" sections of CROCODILE's views.

4.2. Equant router tool

Information on the router security tool developed by *Equant* [4] is scarce[3], and the tool — presumably named `conf_validation` — does not appear to be available to the

1. Instead of a pattern (a Perl regular expression in RAT), a so-called callout (an arbitrary Perl function) may be specified in order to make a `PASS/FAIL` decision. However, the RAT distribution contains only rudimentary callouts, and the callout interface is not meant for user customizations. Even callouts cannot produce any analysis results beyond the binary `PASS/FAIL` decision.
2. To apply the rule set of the RAT 2.0 distribution, we used our RAT emulation module, see below
3. We tried to contact the authors but received no reply.

public. The Equant tool offers numerous small-scale tests comparable in scope to the capabilities of CROCODILE's `CompoundPatterns` module, equipped with only basic patterns; however, its most striking feature seems to be the analysis of the interplay of the rules of an ACL. In this respect, the tool surpasses the single-lines analysis of RAT: Within the rules of each ACL, the tool compares all possible pairs of rules and checks if they are contradictory, or if one of them is redundant. However, the complete whiteset and blackset resulting from the combined effect of *all* rules is never computed; accordingly, most of the security checks described in Section 3.5 are not available.

With R denoting the set of packets matching an ACL rule r, the tool recognizes a rule r_i as *redundant* relative to a rule r_j, if $R_i \subseteq R_j$ and both rules are of the same type ('permit' or 'deny', respectively). Such a rule is not directly harmful, but should be removed for the sake of clarity. Note, however, that only those rules are recognized as redundant whose effect is a sub-effect of only one other rule in this ACL. That is, if the effect of a rule is implied only by the combined effect of two or more other rules, the redundancy is not recognized. In contrast, our approach yields the complete set of redundant rules.

Rules r_i, r_j (i < j) are recognized as *inconsistent* if $R_j \subseteq R_i$ and one of them is a "`deny`", the other a "`permit`" rule. In this case, the net effect of r_j is void, probably contrary to the intention of its author. CROCODILE recognizes this, too, and flags such rules as dead.

4.3. Firewall Analysis Tools

Firewall analysis tools analyze ACLs and routing tables, which is only one of the many facets of CROCODILE.

The *Lumeta Firewall Analyzer* (LFA) [7] simulates a firewall's filtering behavior for all possible IP packets and produces HTML output showing what packets from and to what hosts and services this firewall configuration will or will not let pass. In this respect, LFA's processing of firewall rules is very similar to CROCODILE's processing of a router ACL. The LFA output offers the passed and dropped packets indexed by service and by host (or host group), separated in inbound and outbound direction. This is conceptually rather similar to the functionality of CROCODILE, though more conveniently presented.

LFA works by translating configuration files for firewalls from vendor-specific format to a common intermediate configuration language, and also draws on the firewall's routing table in order to determine what subnets are behind the firewall's network interfaces (this is comparable to CROCODILE's `Connectivity` module). While CROCODILE explicitly computes the blackset and whiteset of an ACL, LFA seems to build its picture of the filtering behavior by successively probing individual combinations of source, destination, and service (which may in turn contain wildcards). However, dead rules (as in CROCODILE) cannot be discovered in this way. Unfortunately, a detailed comparison of the two approaches, in particular regarding performance, coverage, and generality, is not possible based on the information currently available. The public availability of the LFA tool is unclear — it is mentioned, but not marketed in any way on the Lumeta web site.

Another tool for ACL analysis is based on *translating ACL rules to logical clauses in a Prolog-like language*, which are then used by an inference engine to answer questions about the filtering effect of this ACL [8]. The clauses gained from the ACLs are enriched by general logical clauses encoding typical routing concepts such as the accessibility of certain subnets or services. Queries may also be posed using these additional concepts. Users may even program their own logical clauses, which makes the tool extensible in a way similar to CROCODILE.

Even if the tool seems rather immature on the whole and there is apparently no experience with complex real-world ACLs yet, the approach is nevertheless rather interesting conceptually. Encoding application-specific rules in a Prolog-like language may be more natural for certain problems than using Perl; however, it is also less flexible, as the rules no longer operate on the original router configuration description, but on the predefined set of logical atoms. The performance of such an inference-based approach seems critical; the authors make no statements about this, and they have apparently tested their tool with only relatively simple ACLs (10–20 rules) so far. The dissection of overlapping ACLs rules mentioned only in passing raises additional concerns about the performance of this approach.

Another tool to interactively query the filtering effect of an ACL is presented in [9], which again answers questions about accepted and rejected IP packets. Redundant rules are flagged, though not inconsistent ones. Furthermore, the differential blackset and whiteset of two versions of an ACL may be computed. Extending the tool by higher-level concepts as is possible with [8] or CROCODILE is not supported. The bit vector representation used for ACLs makes formulating queries and interpreting results rather cumbersome; this approach may be more appropriate for a router compiling ACL rules to a more efficiently evaluated form than for a human user, who thinks rather at the level of subnets and services than at the level of bit vectors.

5. Conclusions and Outlook

CROCODILE is an easily extended and flexible tool supporting the human expert in the security analysis of router configurations. It relieves the user of cumbersome details and grades all its findings according to their severity while

adding many other useful pieces of information. The tool presents a wealth of information in various topic-specific views.

We have analyzed various practical configurations, originating, however, from only two external user organizations (both with trained staff) so far. CROCODILE has found irregularities in most of these configurations — frequently even grave errors such as references to undefined ACLs! Configuration defects thus seem to be the norm rather than the exception, and security breaches often remain undiscovered for long times. The fact that the checker modules existing so far have uncovered many problems overlooked even by trained staff underlines the validity of our approach.

CROCODILE offers truly novel functionality in various places, particularly regarding the features building on the global view of the router configuration and on the blackset and whiteset analysis. Nevertheless, in spite of all its functionality, CROCODILE cannot replace the human expert — the mere customization of the configurable checker modules requires context knowledge that is not available to a tool. Interpreting all details of the evaluation results will always require the expertise of a networking specialist. Fully automated corrections were thus never a goal of CROCODILE.

The development of CROCODILE continues, including the creation of new checker modules for additional evaluation areas. Further development will be driven by user demand. However, a large, but fixed inventory of checker modules seems less important than the simple extensibility of the CROCODILE framework enabled by the framework's inheritance mechanisms. CROCODILE's application programmer interface provides substantial support for the further extension of checking capabilities, most prominently

- a clean "plug in" interface for new checker modules, supplying each module with only the relevant pieces of information, based on BNF pattern specifications;
- an internal database for storing findings, cross references, fix suggestions, and other useful information in structured format, with a simple yet versatile interface for checker modules;
- a hypertext report generator that presents the contents of the database in uniform style and groups related findings as logical views connected via hyperlinks;
- a differential display mode that automatically applies to any (new) module — including the RAT emulation module.

By defining suitable patterns and handlers, the CROCODILE approach may even be extended to the security checking of other text-based configuration descriptions — ranging from routers by other vendors over firewalls to general network services.

Information about the current status of CROCODILE is available on the homepage at `www.iese.fraunhofer.de/crocodile`, which includes an online demonstration and a detailed manual in German and English version.

6. References

[1] S. Groß, R. Schwarz, *Ein Werkzeug zur Sicherheitsüberprüfung von Cisco IOS Routerkonfigurationen,* IESE Report Nr. 078/01D, Fraunhofer IESE, Kaiserslautern, Germany, December 2001.

[2] Cisco Systems Inc., *Cisco IOS 12.0 Configuration Fundamentals*, Cisco Press, 1999.

[3] Center for Internet Security, *Router Audit Tool,* March 2003. `http://www.cisecurity.org`

[4] D. Valois, C. Llorens, "Identification of Security Holes in Router Configurations", *Proceedings of the 14th Annual Computer Security Incident Handling Conference (FIRST 2002)*,Hilton Waikoloa Village, Hawaii, June 24-28, 2002. `http://www.first.org/events/progconf/2002/d4-04-valois-paper.pdf`

[5] National Security Agency: *Router Security Configuration Guide*. February 2003. `http://www.nsa.gov/snac/cisco`

[6] P. Ferguson, D. Senie, *Network Ingress Filtering – Defeating Denial of Service Attacks which employ IP Source Address Spoofing*, RFC 2827, The Internet Society, May 2000. `http://www.ietf.org/rfc/rfc2827.txt`

[7] A. Wool, "Architecting the Lumeta Firewall Analyzer", *Proceedings of the 10th USENIX Security Symposium*, Washington, D.C., August 13-17, 2001 `http://www.usenix.org/events/sec01/wool.html`

[8] P. Eronen, J. Zitting, An expert system for analyzing firewall rules, *Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec2001)*, Copenhagen, Denmark, 2001. `http://www.cs.hut.fi/~peronen/publications`

[9] S. Hazelhurst, *Algorithms for Analysing Firewall and Router Access Lists*, Tech. Report TR-Wits-CS-99-5, University of the Witwatersrand, South Africa, July, 1999. `http://citeseer.nj.nec.com/hazelhurst99algorithm.html`